Hot code is faster code

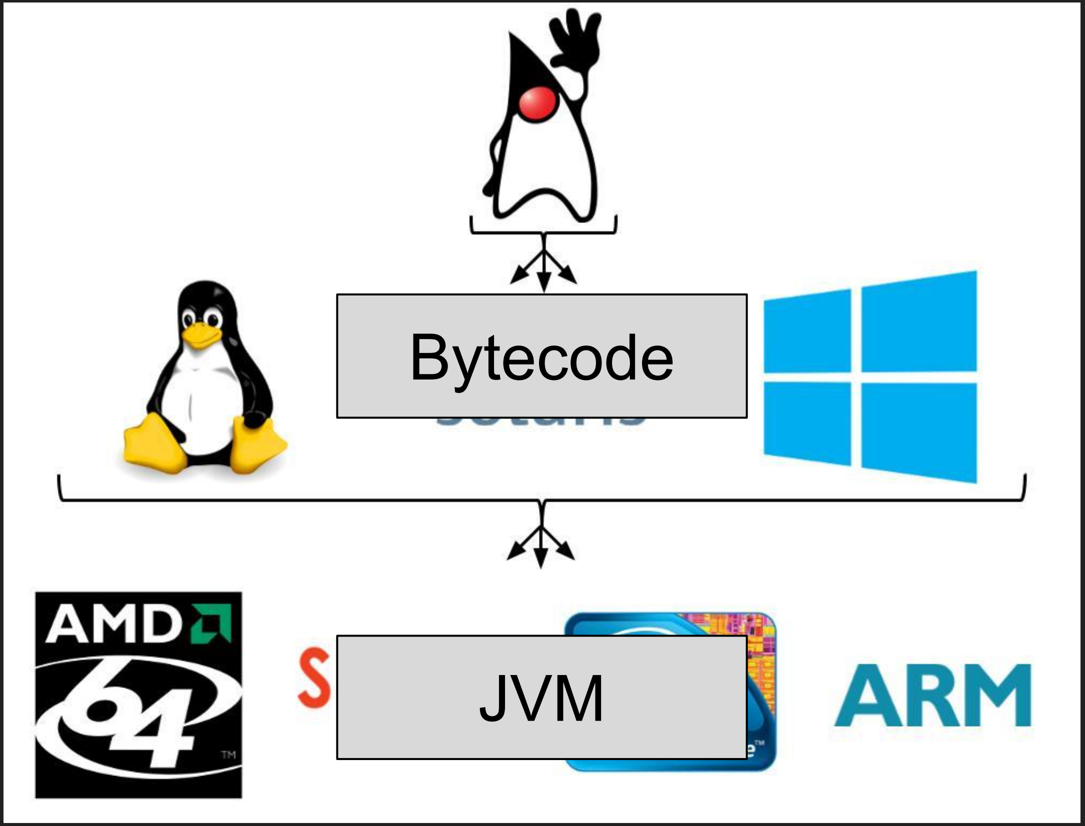
Addressing JVM warm-up

Mark Price
LMAX Exchange

The JVM warm-up problem?

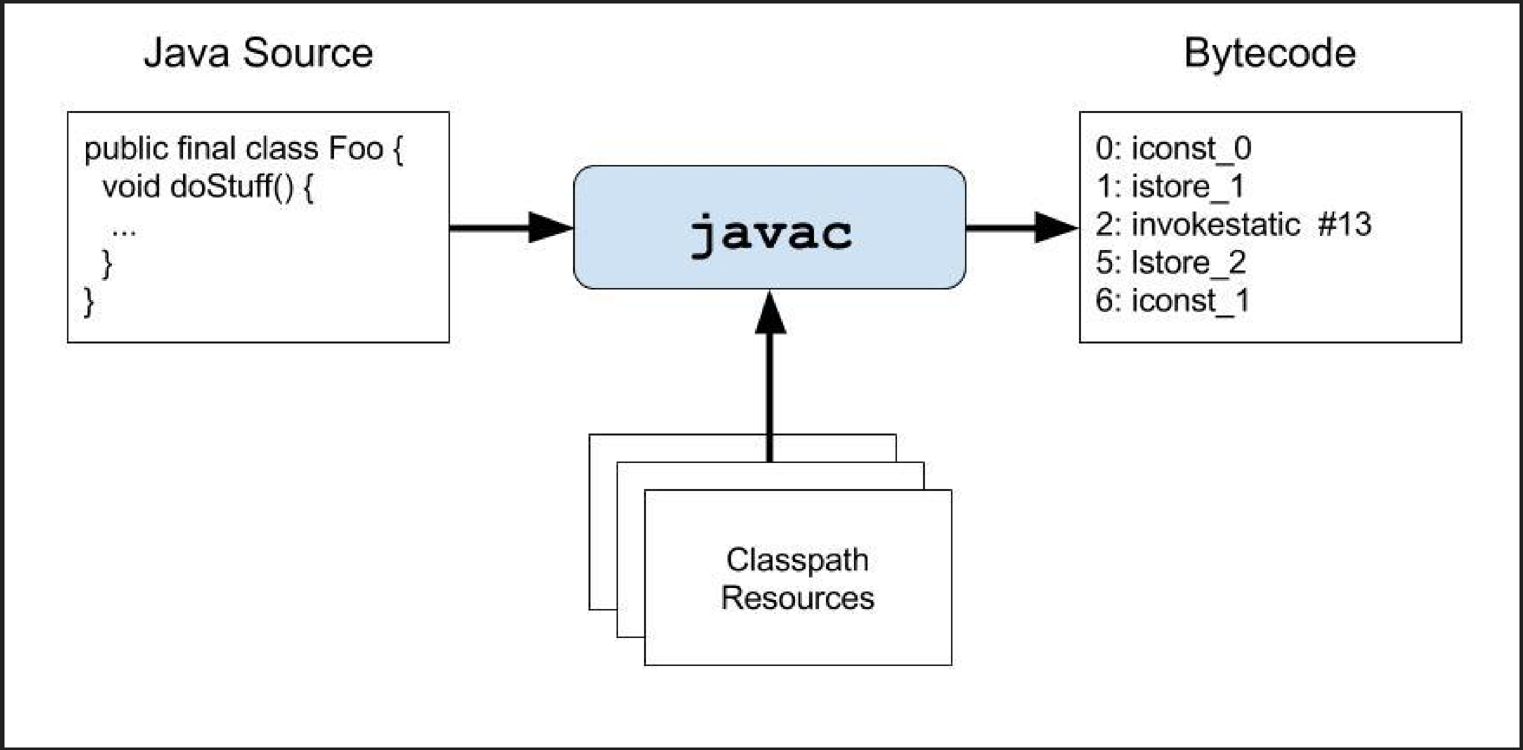
The JVM warm-up feature!

In the beginning



Images from Wikipedia

What does the JVM run?



```
public final class Foo {
  void doStuff() {
   ...
  }
}
```

```
0: iconst_0
1: istore_1
2: invokestatic  #13
5: lstore_2
6: iconst_1
```

THE INTERPRETER

An example (source)

```
public static int doLoop10()
{
    int sum = 0;
    for(int i = 0; i < 10; i++)
    {
        sum += i;
    }

    return sum;
}
```

An example (decompiling)

```
$JAVA_HOME/bin/javap
    -p  // show all classes and members
    -c  // disassemble the code
    -cp $CLASSPATH
com.epickrram.talk.warmup.example.loop.FixedLoopCount
```

An example (bytecode)

```
 0: iconst_0                    // load ‘0’ onto the stack
 1: istore_0                    // store top of stack to #0 (sum)
 2: iconst_0                    // load ‘0’ onto the stack
 3: istore_1                    // store top of stack to #1 (i)
 4: iload_1                     // load value of #1 onto stack
 5: bipush        10            // push ‘10’ onto stack
 7: if_icmpge     20            // compare stack values, jump to 20 if #1 >= 10
10: iload_0                     // load value of #0 (sum) onto stack
11: iload_1                     // load value of #1 (i) onto stack
12: iadd                        // add stack values
13: istore_0                    // store result to #0 (sum)
14: iinc          1, 1          // increment #1 (i) by 1
17: goto          4             // goto 4
20: iload_0                     // load value of #0 (sum) onto stack
21: ireturn                     // return top of stack
```

https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

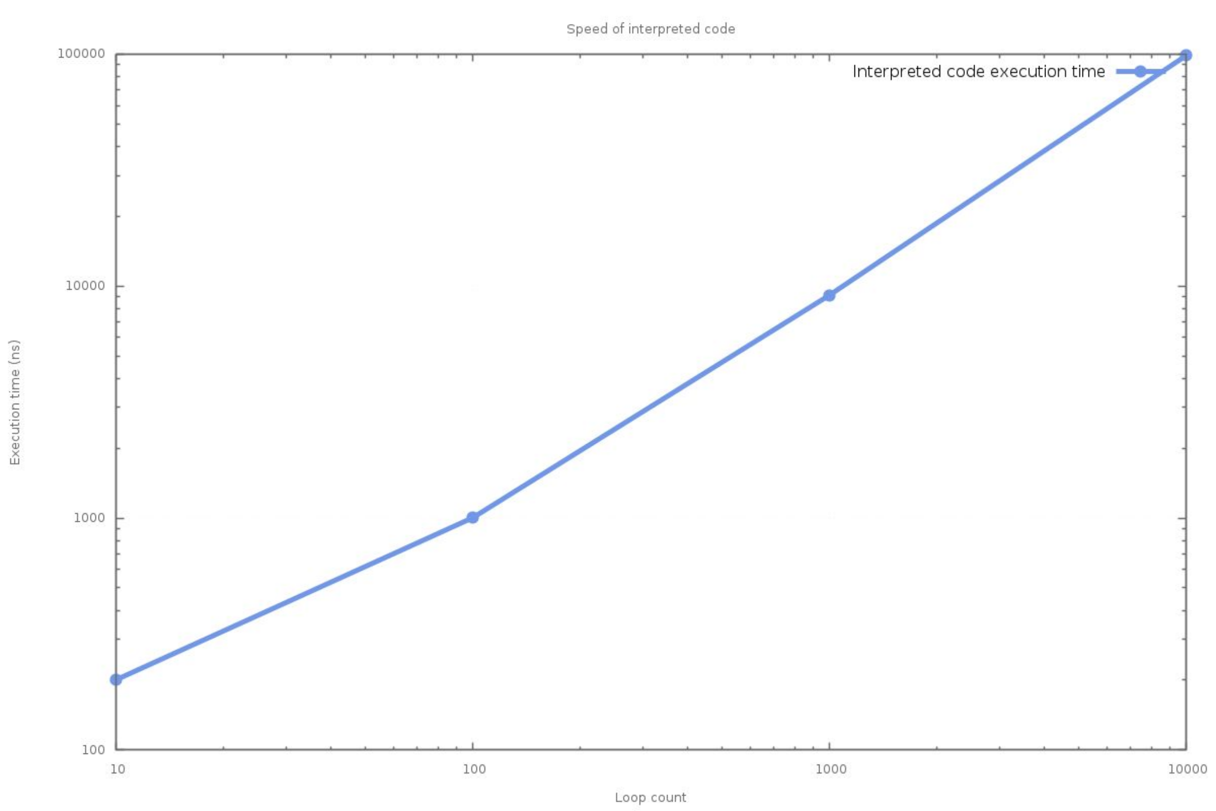
Interpreted mode

- Each bytecode is interpreted and executed at runtime
- Start up behaviour for most JVMs
- A runtime flag can be used to force interpreted mode
- -Xint
- No compiler optimisation performed

Speed of interpreted code

```
@Benchmark
public long fixedLoopCount10()
{
   return FixedLoopCount.doLoop10();
}

@Benchmark
public long fixedLoopCount100()
{
   return FixedLoopCount.doLoop100();
}
...
```

Speed of interpreted code



count	time
x10	0.2 us
x100	1.0 us
x1000	9.1 us
x10000	98.5 us

doLoop10

doLoop100

doLoop1000

doLoop10000

THE COMPILER

Enter the JIT

- Just In Time, or at least, deferred
- Added way back in JDK 1.3 to improve performance
- Replaces interpreted code with optimised machine code
- Compilation happens on a background thread
- Monitors running code using counters
- Method entry points, loop back-edges, branches

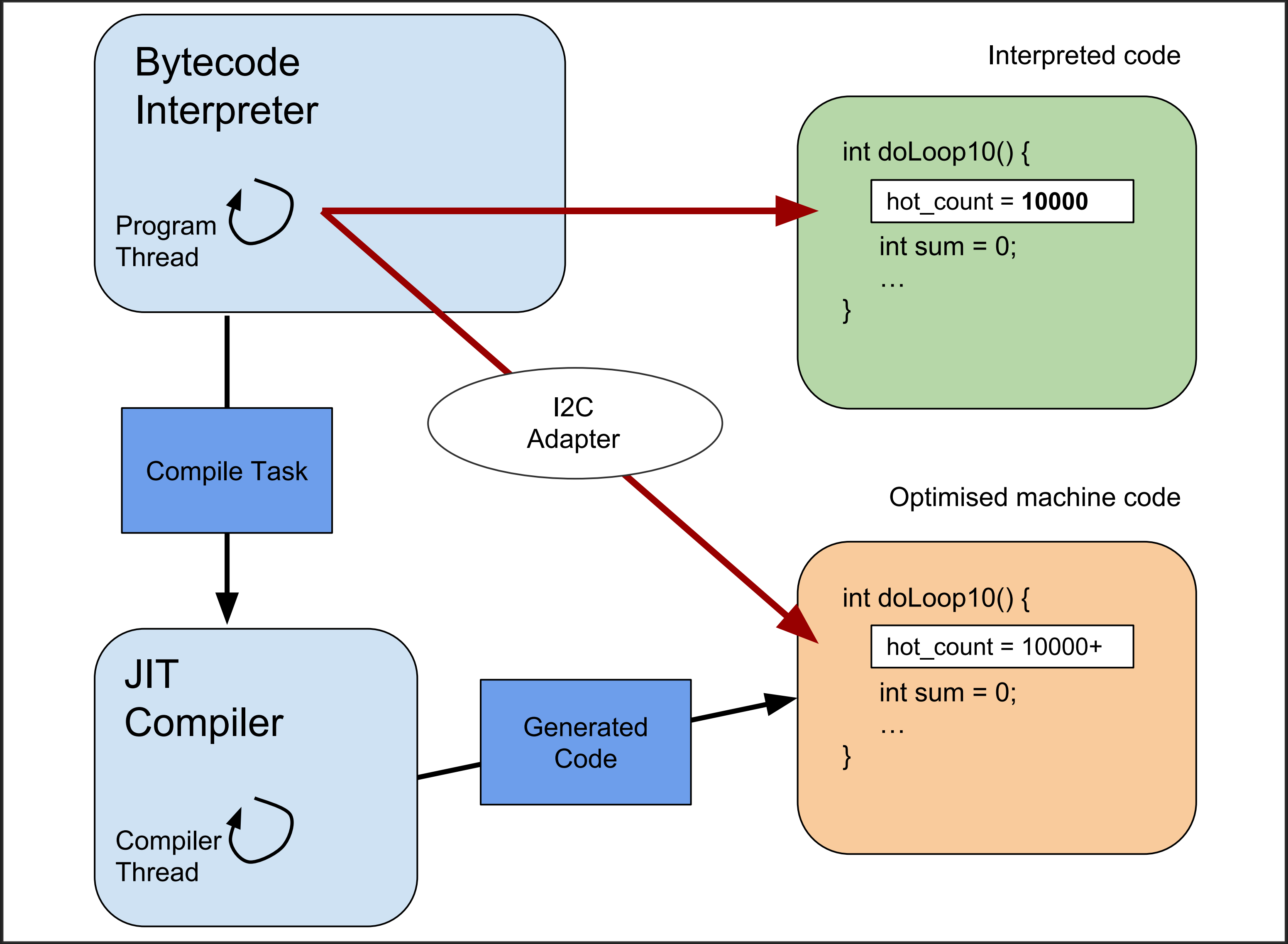
Interpreter Counters

```
public static int doLoop10()
{ // method entry point
    int sum = 0;
    for(int i = 0; i < 10; i++)
    {
        sum += i;

        // loop back-edge
    }
    return sum;
}
```

Two flavours

- Client (C1) [-client]
- Server (C2) [-server]
- Client is focussed on desktop/GUI targeting fast start-up times
- Server is aimed at long-running processes for max performance
- *-server* should produce most optimised code
- 64-bit JDK ignores *-client* and goes straight to *-server*
- -XX:+TieredCompilation (default)

Compiler Operation



```
int doLoop10() {
    hot_count = 10000
    int sum = 0;
    …
}
```

```
int doLoop10() {
    hot_count = 10000+
    int sum = 0;
    …
}
```

LOOKING CLOSER

Steps to unlock the secrets of the JIT

1. -XX:+UnlockDiagnosticVMOptions
2. -XX:+LogCompilation
3. Run program
4. View hotspot_pid<pid>.log
5. *facepalm*

TMI

```
<task_queued compile_id='15' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='205' backedge_count='2048' iicount='205' level='3' stamp='0.096'
comment='tiered' hot_count='205'/>
<writer thread='140617399015168'/>
<nmethod compile_id='15' compiler='C1' level='3' entry='0x00007fe4612b5080' size='1008' address='0x00007fe4612b4f10' relocation_offset='296' insts_offset='368' stub_offset='720'
scopes_data_offset='880' scopes_pcs_offset='920' dependencies_offset='1000' oops_offset='864' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='2793'
backedge_count='27950' iicount='2799' stamp='0.097'/>
<writer thread='140619223398144'/>
<task_queued compile_id='16' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='3456' backedge_count='34550' iicount='3456' stamp='0.097'
comment='tiered' hot_count='3456'/>
<writer thread='140617407436544'/>
<nmethod compile_id='16' compiler='C2' level='4' entry='0x00007fe4612b8080' size='448' address='0x00007fe4612b7f50' relocation_offset='296' insts_offset='304' stub_offset='368'
scopes_data_offset='400' scopes_pcs_offset='408' dependencies_offset='440' oops_offset='392' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='22758'
backedge_count='227698' iicount='22783' stamp='0.099'/>
<make_not_entrant thread='140617407436544' compile_id='15' compiler='C1' level='3' stamp='0.099'/>
<writer thread='140619223398144'/>
<task_queued compile_id='17' compile_kind='osr' method='com/epickrram/talk/warmup/example/loop/FixedLoopCountMain main ([Ljava/lang/String;)V' bytes='13' count='1' backedge_count='60416'
iicount='1' osr_bci='0' level='3' stamp='0.100' comment='tiered' hot_count='60416'/>
<writer thread='140617402173184'/>
<nmethod compile_id='17' compile_kind='osr' compiler='C1' level='3' entry='0x00007fe4612b7b20' size='1440' address='0x00007fe4612b7990' relocation_offset='296' insts_offset='400'
stub_offset='1040' scopes_data_offset='1208' scopes_pcs_offset='1304' dependencies_offset='1432' oops_offset='1184' method='com/epickrram/talk/warmup/example/loop/FixedLoopCountMain main
([Ljava/lang/String;)V' bytes='13' count='1' backedge_count='83294' iicount='1' stamp='0.101'/>
<writer thread='140619223398144'/>
<task_queued compile_id='18' method='com/epickrram/talk/warmup/example/loop/FixedLoopCountMain main ([Ljava/lang/String;)V' bytes='13' count='1' backedge_count='84305' iicount='1' level='3'
stamp='0.101' comment='tiered' hot_count='1'/>
<task_queued compile_id='19' compile_kind='osr' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='23321' backedge_count='233206' iicount='23321'
osr_bci='4' stamp='0.101' comment='tiered' hot_count='233206'/>
<writer thread='140617402173184'/>
<nmethod compile_id='18' compiler='C1' level='3' entry='0x00007fe4612b7560' size='1408' address='0x00007fe4612b73d0' relocation_offset='296' insts_offset='400' stub_offset='1008'
scopes_data_offset='1176' scopes_pcs_offset='1272' dependencies_offset='1400' oops_offset='1152' method='com/epickrram/talk/warmup/example/loop/FixedLoopCountMain main ([Ljava/lang/String;)V'
bytes='13' count='1' backedge_count='94126' iicount='1' stamp='0.101'/>
<writer thread='140619223398144'/>
<task_queued compile_id='20' compile_kind='osr' method='com/epickrram/talk/warmup/example/loop/FixedLoopCountMain main ([Ljava/lang/String;)V' bytes='13' count='1' backedge_count='108881'
iicount='1' osr_bci='0' stamp='0.102' comment='tiered' hot_count='108881'/>
<writer thread='140617409541888'/>
<nmethod compile_id='19' compile_kind='osr' compiler='C2' level='4' entry='0x00007fe4612b5da0' size='608' address='0x00007fe4612b5c50' relocation_offset='296' insts_offset='336'
stub_offset='528' scopes_data_offset='560' scopes_pcs_offset='568' dependencies_offset='600' oops_offset='552' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I'
bytes='22' count='70199' backedge_count='702134' iicount='70232' stamp='0.103'/>
```

Tiered Compilation in action

```
# cat hotspot_pid14969.log | grep "FixedLoopCount doLoop10 ()I"

<task_queued compile_id='15'

method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I'

bytes='22' count='205' backedge_count='2048' iicount='205'

level='3' stamp='0.096' comment='tiered' hot_count='205'/>

<nmethod compile_id='15' compiler='C1' level='3' entry='0x00007fe4612b5080' size='1008'
address='0x00007fe4612b4f10' relocation_offset='296' insts_offset='368' stub_offset='720'
scopes_data_offset='880' scopes_pcs_offset='920' dependencies_offset='1000' oops_offset='864'
method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22' count='2793'
backedge_count='27950' iicount='2799' stamp='0.097'/>
```

Tiered Compilation in action

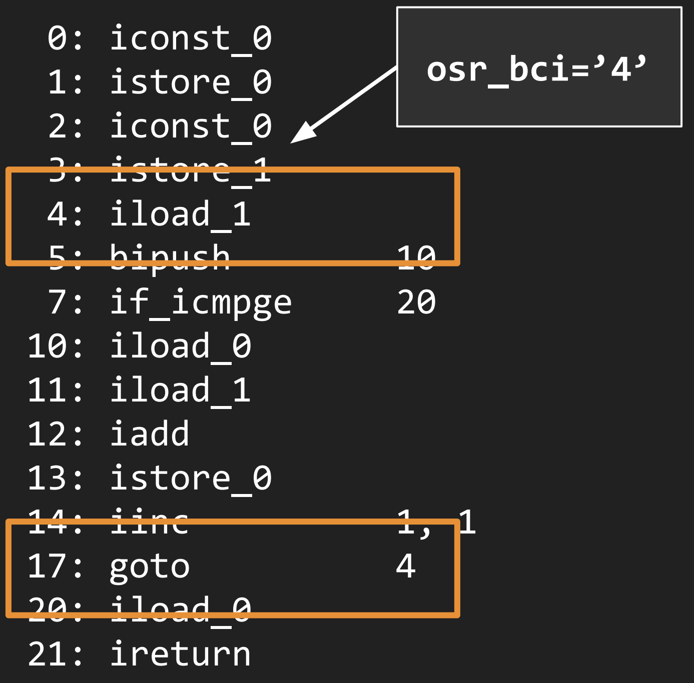
```
<task_queued compile_id='16' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()
I' bytes='22' count='3456' backedge_count='34550' iicount='3456' stamp='0.097' comment='tiered'
hot_count='3456'/>

<nmethod compile_id='16' compiler='C2' level='4' entry='0x00007fe4612b8080' size='448'
address='0x00007fe4612b7f50' relocation_offset='296' insts_offset='304' stub_offset='368'
scopes_data_offset='400' scopes_pcs_offset='408' dependencies_offset='440' oops_offset='392'
method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22'
count='22758' backedge_count='227698' iicount='22783' stamp='0.099'/>
```

Tiered Compilation in action

```
<task_queued compile_id='19' compile_kind='osr'
method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I' bytes='22'
count='23321' backedge_count='233206' iicount='23321' osr_bci='4' stamp='0.101' comment='tiered'
hot_count='233206'/>
```

```
<nmethod compile_id='19' compile_kind='osr' compiler='C2' level='4' entry='0x00007fe4612b5da0'
size='608' address='0x00007fe4612b5c50' relocation_offset='296' insts_offset='336'
stub_offset='528' scopes_data_offset='560' scopes_pcs_offset='568' dependencies_offset='600'
oops_offset='552' method='com/epickrram/talk/warmup/example/loop/FixedLoopCount doLoop10 ()I'
bytes='22' count='70199' backedge_count='702134' iicount='70232' stamp='0.103'/>
```

Tiered Compilation in action

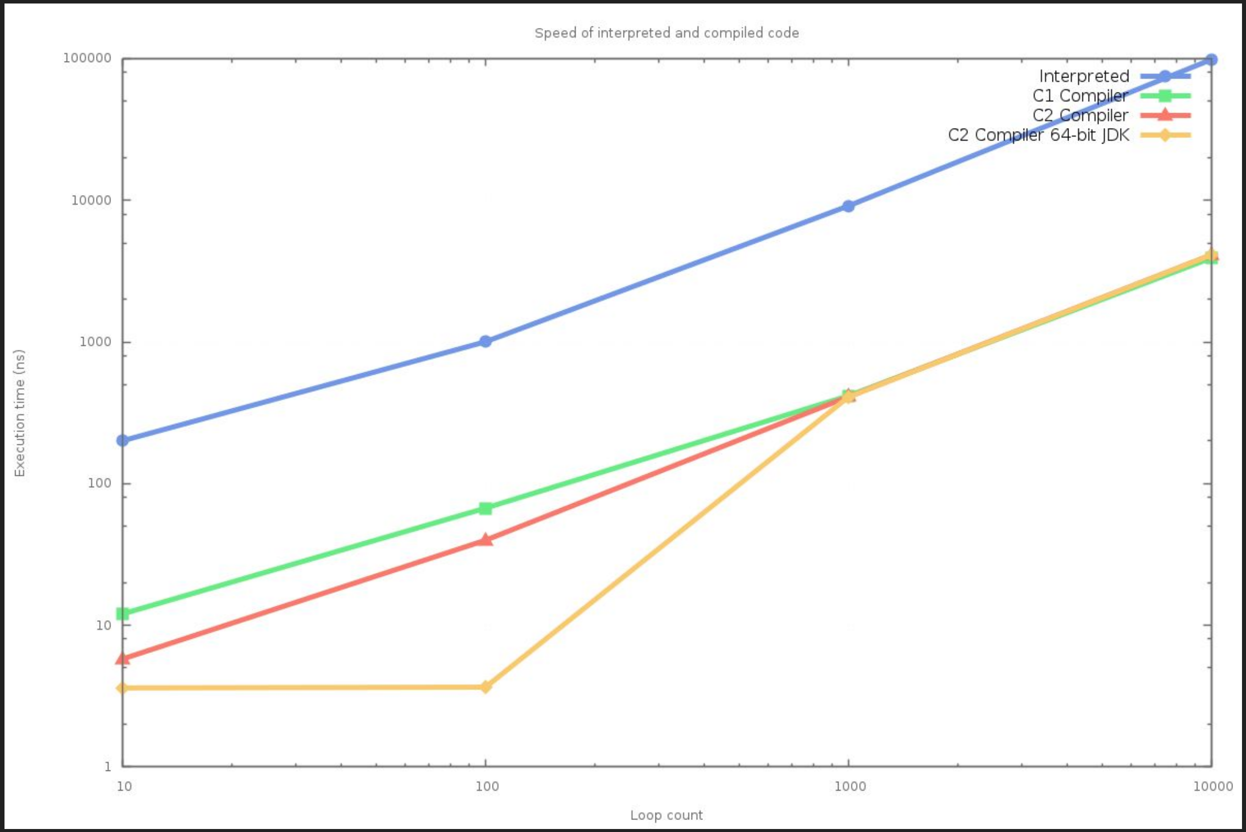


osr_bci=’4’

```
 0: iconst_0
 1: istore_0
 2: iconst_0
 3: istore_1
 4: iload_1
 5: bipush        10
 7: if_icmpge     20
10: iload_0
11: iload_1
12: iadd
13: istore_0
14: iinc          1, 1
17: goto          4
20: iload_0
21: ireturn
```

- Method execution starts in interpreted mode
- C1 compilation after back-edge count > C1 threshold
- C2 compilation after back-edge count > C2 threshold
- OSR starts executing compiled code before the loop completes

Compiler comparison



> 20x speed up

Speed up will be much greater for more complex methods and method hierarchies (typically x1,000+).

KNOWN UNKNOWNS

Uncommon Traps

- Injected by the compiler into native code
- Detect whether assumptions have been invalidated
- Bail out to interpreter
- Start the compilation cycle again

Example: TypeProfiles

- Virtual method invocation of interface method
- Observe that only one implementation exists
- Optimise virtual call by inlining
- Performance win!
- Spot the assumption

Type Profiles

```
public interface Calculator
{
    int calculateResult(final int input);
}
```

Type Profiles

```
static volatile Calculator calculator = new FirstCalculator();
...
int accumulator = 0;
long loopStart = System.nanoTime();
for(int i = 1; i < 1000000; i++) {
    accumulator += calculator.calculateResult(i);

    if(i % 1000 == 0 && i != 0) {
        logDuration(loopStart);
        loopStart = System.nanoTime();
    }
    ITERATION_COUNT.lazySet(i);
```

Type Profiles

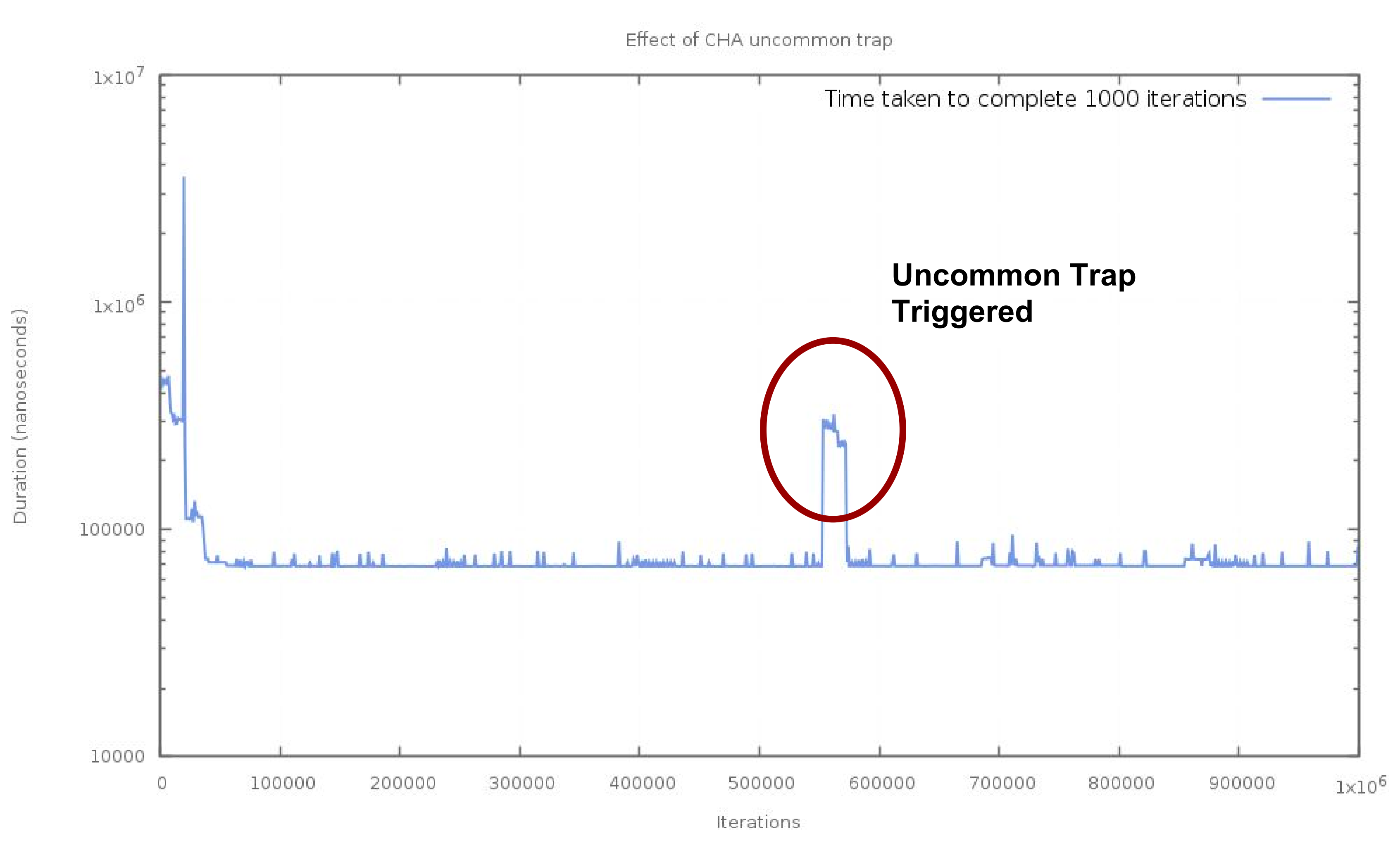
```
// attempt to load another implementation
// will invalidate previous assumption

if(ITERATION_COUNT.get() > 550000 && !changed) {
    calculator = (Calculator)
       Class.forName("....SecondCalculator").newInstance();
```

Type Profiles

```
Loop at 550000 took 69090 ns
Loop at 551000 took 68890 ns
Loop at 552000 took 68925 ns
[Loaded com.epickrram.talk.warmup.example.cha.SecondCalculator ]
Loop at 553000 took 305987 ns
Loop at 554000 took 285183 ns
Loop at 555000 took 281293 ns
…
Loop at 572000 took 237633 ns
Loop at 573000 took 71779 ns
Loop at 574000 took 84552 ns
Loop at 575000 took 69061 ns
```

```
-XX:+TraceClassLoading
```

Type Profiles



Uncommon Trap Triggered

Type Profiles

```
<task compile_id='9' ...

<klass id='822' name='com/epickrram/talk/warmup/example/cha/FirstCalculator'/>

<call virtual='1' inline='1' receiver='822' receiver_count='22321'/>
<uncommon_trap reason='class_check' action='maybe_recompile'
              comment='monomorphic vcall checkcast'/>

...

<uncommon_trap reason='class_check' action='maybe_recompile' compile_id='9'>
   <jvms ... class_check_traps='1'/>
</uncommon_trap>
```

STRATEGIES

What’s wrong with cold code?

- Interpreted code will take time to be compiled
- Compilation itself is fast, but methods may not be ‘hot’
- Compiled code cleared after JVM restart/code release
- E.g. market-open in finance applications
- E.g. auction sites

Strategy #1: Warm-up in-place

- Post release or restart
- Send traffic through the system
- Must match production-like traffic
- Must exercise all latency-sensitive code-paths
- Requires good data isolation

Strategy #1: Understand what is happening

To get this method to work, pay attention to the following:

- Observe (logging)
- Understand (count threshold, size threshold)
- Modify
- GOTO 10
- Then tweak your traffic generator until the desired result is achieved

Strategy #2: Ahead-of-time Compile

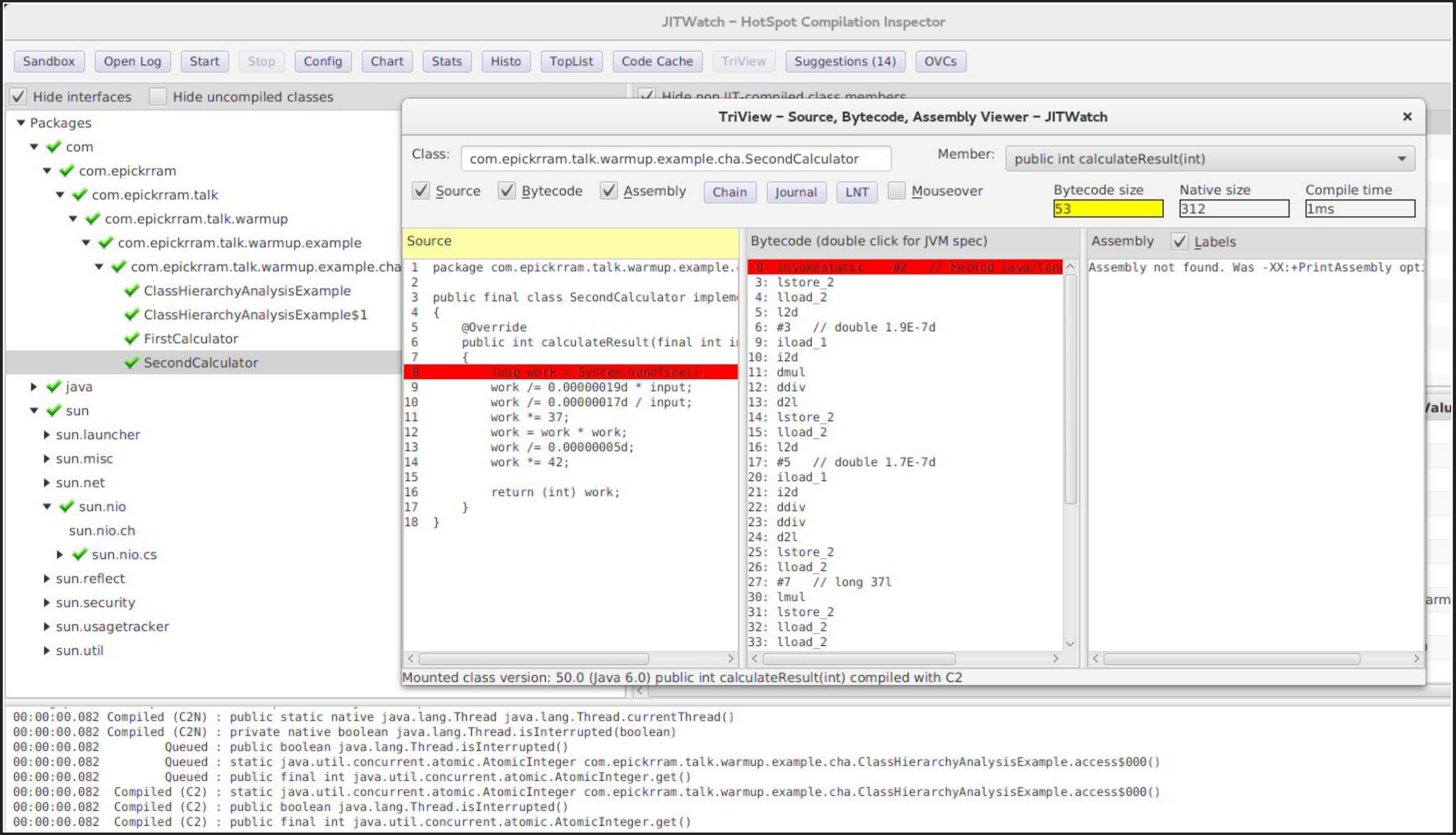
- Commercial solutions available
- Compile user bytecode into machine code
- Pre-compiled runtime
- No warm-up effects
- No profile-guided optimisation

Strategy #3: Zing ReadyNow!

- Commercial solution
- Compiler profile is logged to file
- On startup, if log file exists, compilation is performed up-front
- Greatly reduces warm-up time
- Susceptible to large code refactorings
- Ship compiler profile from perf-test environment to production

A FEW TOOLS

JITWatch



Class: com.epickrram.talk.warmup.example.cha.SecondCalculator

Member: public int calculateResult(int)

Bytecode size: 53

Native size: 312

Compile time: 1ms

```
package com.epickrram.talk.warmup.example.

public final class SecondCalculator impleme
{
    @Override
    public int calculateResult(final int i
    {
        long work = System.nanoTime();
        work /= 0.00000019d * input;
        work /= 0.00000017d / input;
        work *= 37;
        work = work * work;
        work /= 0.00000005d;
        work *= 42;

        return (int) work;
    }
}
```

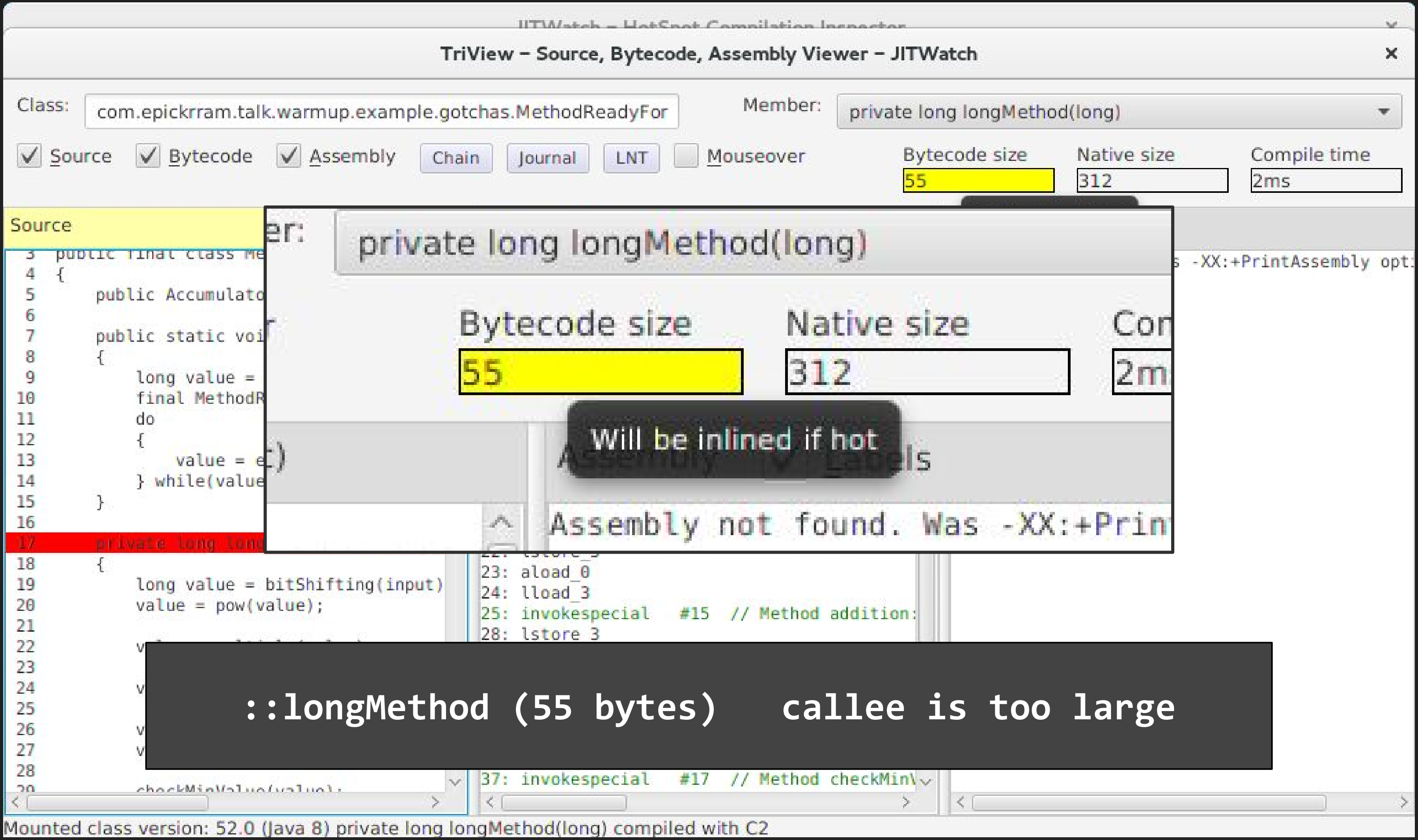
```
 0: invokestatic    #2    // Method java/lan
 3: lstore_2
 4: lload_2
 5: l2d
 6: #3    // double 1.9E-7d
 9: iload_1
10: i2d
11: dmul
12: ddiv
13: d2l
14: lstore_2
15: lload_2
16: l2d
17: #5    // double 1.7E-7d
20: iload_1
21: i2d
22: ddiv
23: ddiv
24: d2l
25: lstore_2
26: lload_2
27: #7    // long 37l
30: lmul
31: lstore_2
32: lload_2
33: lload_2
```

```
Assembly not found. Was -XX:+PrintAssembly opt
```

Mounted class version: 50.0 (Java 6.0) public int calculateResult(int) compiled with C2

```
00:00:00.082 Compiled (C2N) : public static native java.lang.Thread java.lang.Thread.currentThread()
00:00:00.082 Compiled (C2N) : private native boolean java.lang.Thread.isInterrupted(boolean)
00:00:00.082         Queued : public boolean java.lang.Thread.isInterrupted()
00:00:00.082         Queued : static java.util.concurrent.atomic.AtomicInteger com.epickrram.talk.warmup.example.cha.ClassHierarchyAnalysisExample.access$000()
00:00:00.082         Queued : public final int java.util.concurrent.atomic.AtomicInteger.get()
00:00:00.082  Compiled (C2) : static java.util.concurrent.atomic.AtomicInteger com.epickrram.talk.warmup.example.cha.ClassHierarchyAnalysisExample.access$000()
00:00:00.082  Compiled (C2) : public boolean java.lang.Thread.isInterrupted()
00:00:00.082  Compiled (C2) : public final int java.util.concurrent.atomic.AtomicInteger.get()
```

Failure to inline



TriView – Source, Bytecode, Assembly Viewer – JITWatch

Class: com.epickrram.talk.warmup.example.gotchas.MethodReadyFor

Member: private long longMethod(long)

☑ Source ☑ Bytecode ☑ Assembly

□ Mouseover

Bytecode size: 55

Native size: 312

Compile time: 2ms

Source

```
public final class Me
{
    public Accumulato

    public static voi
    {
        long value =
        final MethodR
        do
        {
            value =
        } while(value
    }

    private long long
    {
        long value = bitShifting(input)
        value = pow(value);

        v

        v

        v
        v

        checkMinValue(value);
```

```
23: aload_0
24: lload_3
25: invokespecial   #15  // Method addition:
28: lstore_3
37: invokespecial   #17  // Method checkMinV
```

-XX:+PrintAssembly opt

private long longMethod(long)

Bytecode size: 55

Native size: 312

2m

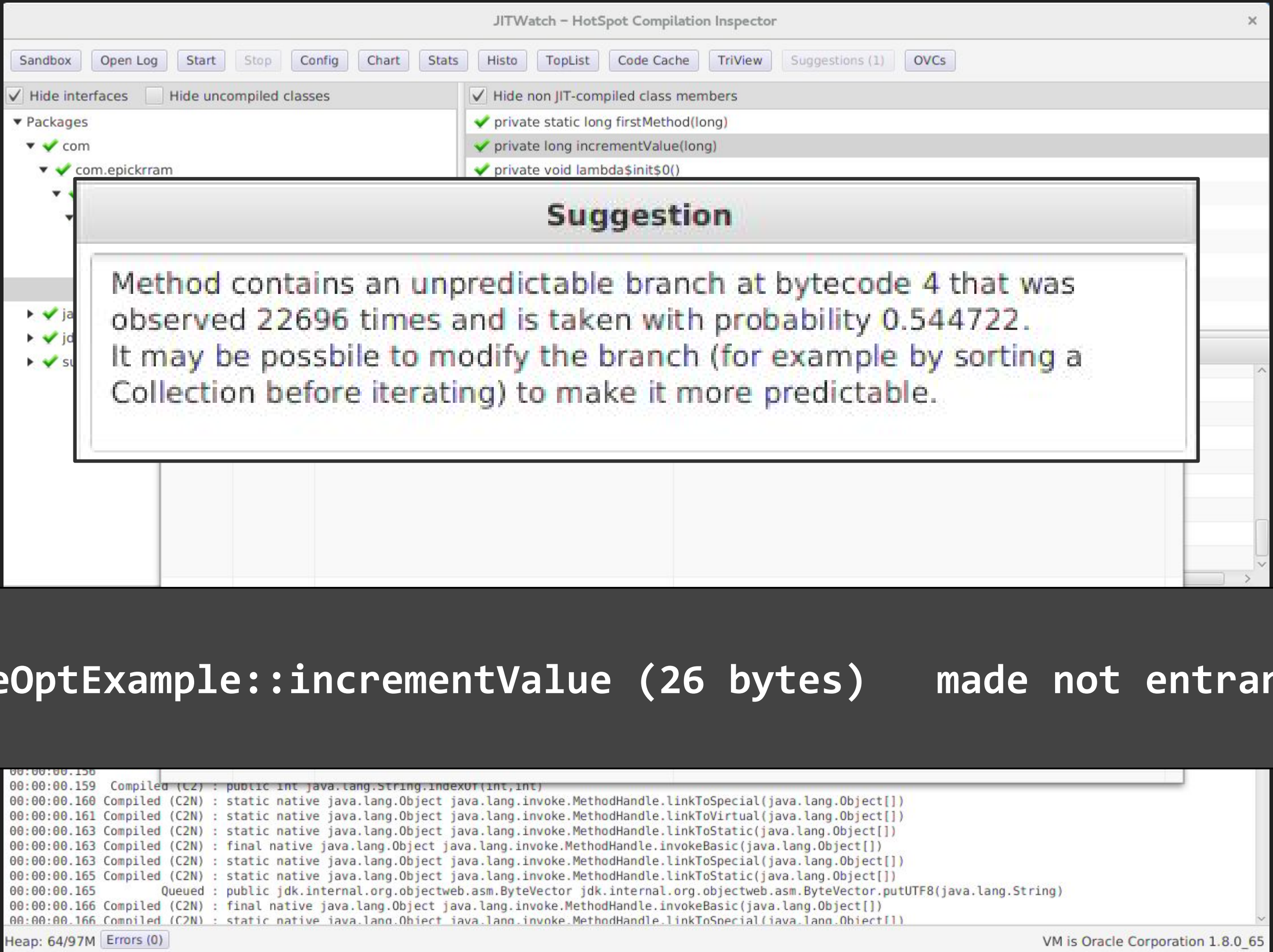
Will be inlined if hot

Assembly not found. Was -XX:+Prin

Mounted class version: 52.0 (Java 8) private long longMethod(long) compiled with C2

```
::longMethod (55 bytes)   callee is too large
```

De-opts



✓ private static long firstMethod(long)
✓ private long incrementValue(long)
✓ private void lambda$init$0()

Suggestion

Method contains an unpredictable branch at bytecode 4 that was observed 22696 times and is taken with probability 0.544722.
It may be possbile to modify the branch (for example by sorting a Collection before iterating) to make it more predictable.

```
DeOptExample::incrementValue (26 bytes)   made not entrant
```

```
00:00:00.159  Compiled (C2) : public int java.lang.String.indexOf(int,int)
00:00:00.160 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToSpecial(java.lang.Object[])
00:00:00.161 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToVirtual(java.lang.Object[])
00:00:00.163 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToStatic(java.lang.Object[])
00:00:00.163 Compiled (C2N) : final native java.lang.Object java.lang.invoke.MethodHandle.invokeBasic(java.lang.Object[])
00:00:00.163 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToSpecial(java.lang.Object[])
00:00:00.165 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToStatic(java.lang.Object[])
00:00:00.165         Queued : public jdk.internal.org.objectweb.asm.ByteVector jdk.internal.org.objectweb.asm.ByteVector.putUTF8(java.lang.String)
00:00:00.166 Compiled (C2N) : final native java.lang.Object java.lang.invoke.MethodHandle.invokeBasic(java.lang.Object[])
00:00:00.166 Compiled (C2N) : static native java.lang.Object java.lang.invoke.MethodHandle.linkToSpecial(java.lang.Object[])
```

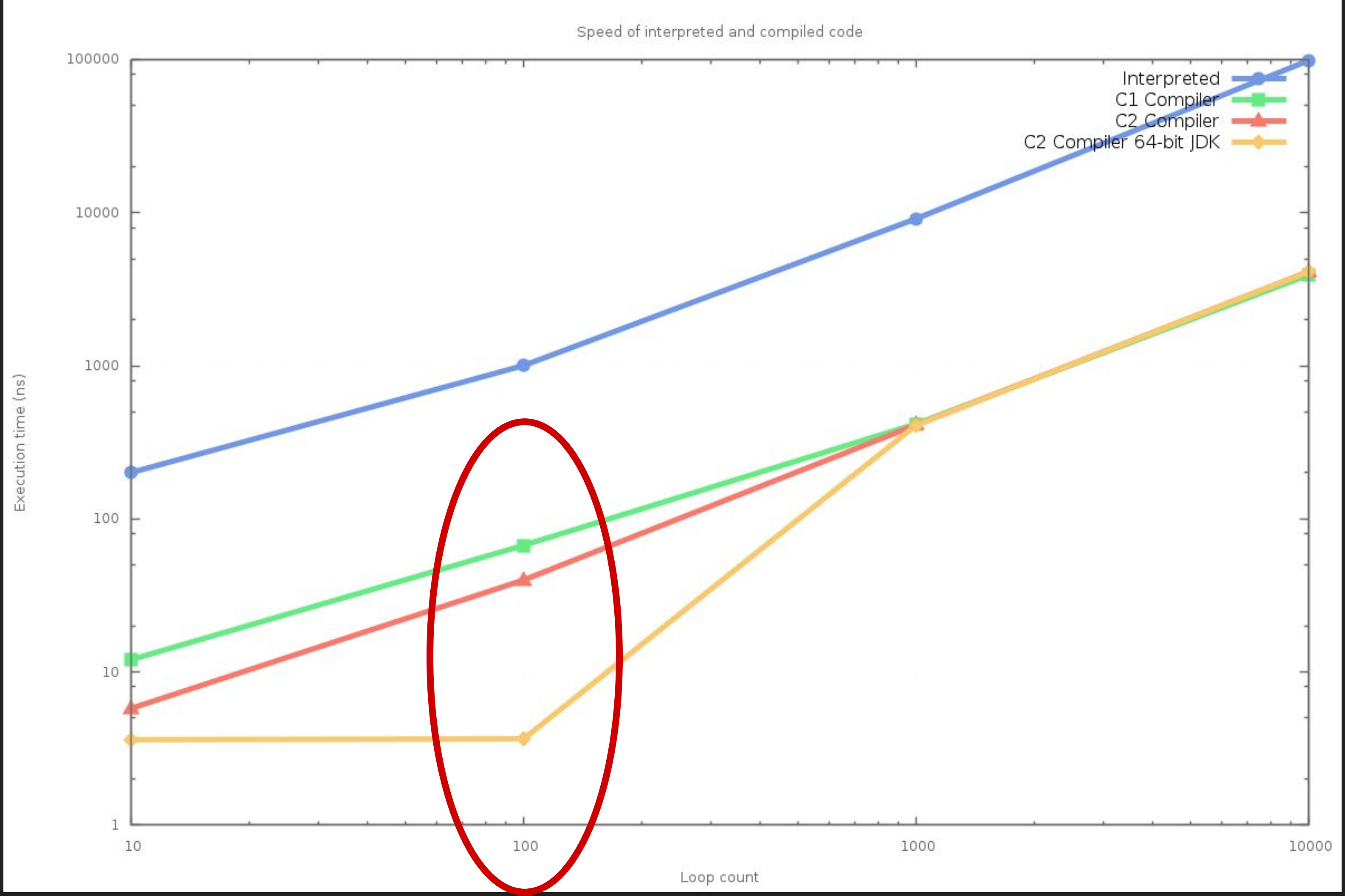
Heap: 64/97M Errors (0)

VM is Oracle Corporation 1.8.0_65

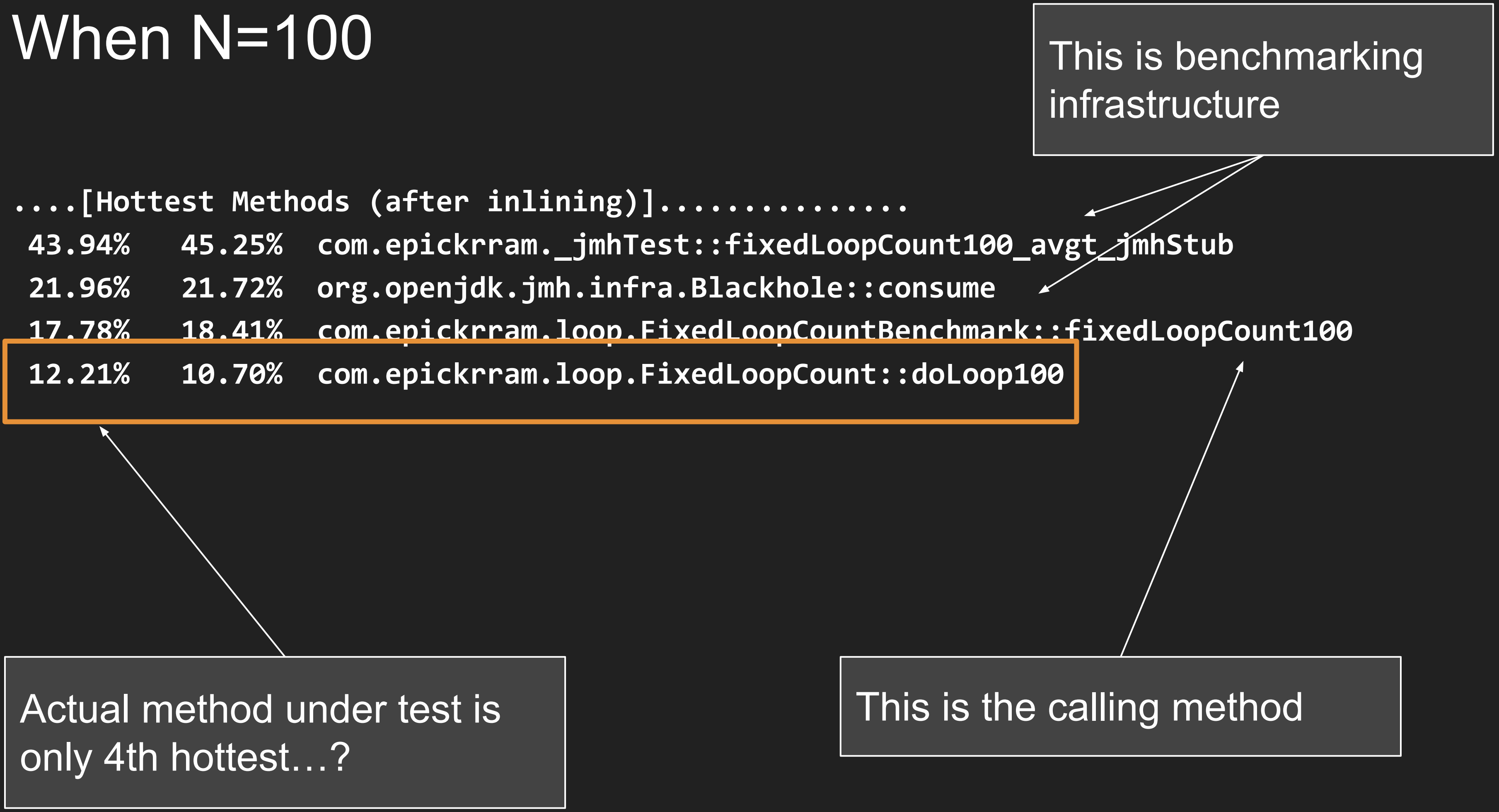
JMH -prof perfasm

- For deep inspection of your code
- Only run within the context of a JMH benchmark
- Uses perf_events (Linux) to sample the thread stack
- Captures -XX:+PrintAssembly
- Matches up executing assembly code with Java methods
- Remember that assembly is arch-specific
- Profile on the same hardware as production systems

Why the difference?



When N=100



This is benchmarking infrastructure

```
....[Hottest Methods (after inlining)]..............
 43.94%   45.25%  com.epickrram._jmhTest::fixedLoopCount100_avgt_jmhStub
 21.96%   21.72%  org.openjdk.jmh.infra.Blackhole::consume
 17.78%   18.41%  com.epickrram.loop.FixedLoopCountBenchmark::fixedLoopCount100
 12.21%   10.70%  com.epickrram.loop.FixedLoopCount::doLoop100
```

Actual method under test is only 4th hottest…?

This is the calling method

When N=100

```
; - com.epickrram.talk.loop.FixedLoopCount::doLoop100@-1 (line 18)
  0.28%    0.30%     0x00007fe1053a3a4c: mov    $0x1356,%eax
  0.49%    0.31%     0x00007fe1053a3a51: add    $0x10,%rsp
  1.01%    0.99%     0x00007fe1053a3a55: pop    %rbp
  4.18%    4.07%     0x00007fe1053a3a56: test   %eax,0x15c215a4(%rip)  {poll_return}
  0.31%    0.13%     0x00007fe1053a3a5c: retq
```

```
sum(0..99) == 4950 == 0x1356
```

Compiler has optimised for-loop into a constant

When N=1000

```
...[Hottest Methods (after inlining)]..........
 94.50%   95.14%  com.epickrram.loop.FixedLoopCount::doLoop1000
  1.57%    1.53%  native_write_msr_safe ([kernel.kallsyms])
  0.54%    0.29%  com.epickrram._jmhTest::fixedLoopCount1000_avgt_jmhStub
  0.26%    0.28%  org.openjdk.jmh.infra.Blackhole::consume
```

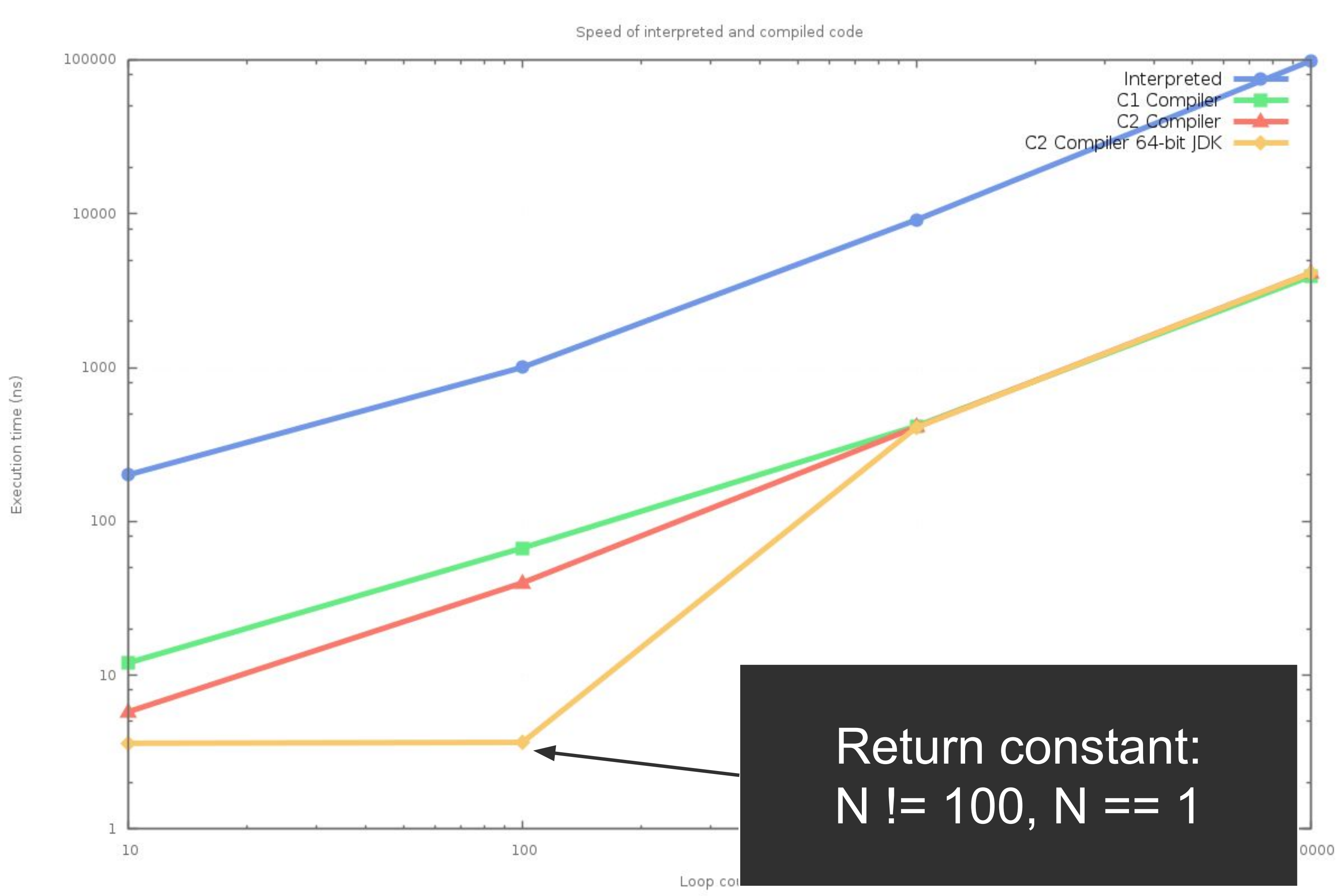
Method under test is now the hottest method

When N=1000

```
                       0x00007f52753a860e: mov    $0x1,%r11d          ;*iload_0
 0.15%    0.46%    ↗   0x00007f52753a8614: add    %r11d,%eax
18.87%   12.60%    │   0x00007f52753a8617: add    %r11d,%eax
18.88%   11.43%    │   0x00007f52753a861a: add    %r11d,%eax
18.88%   45.80%    │   0x00007f52753a861d: add    %r11d,%eax
18.95%   11.87%    │   0x00007f52753a8620: add    $0x6,%eax           ;*iadd
18.28%   12.41%    │   0x00007f52753a8623: add    $0x4,%r11d          ;*iinc
 0.07%    0.14%    │   0x00007f52753a8627: cmp    $0x3e5,%r11d
                   ╰   0x00007f52753a862e: jl     0x00007f52753a8614   ;*if_icmpge
                       0x00007f52753a8630: cmp    $0x3e8,%r11d
                    ╭  0x00007f52753a8637: jge    0x00007f52753a864b
                    │  0x00007f52753a8639: data32 xchg %ax,%ax        ;*iload_0
 0.                                                                    ;*iadd
 0.                                                                    ;*iinc

                                                                a863c   ;*if_icmpge
                    ↘  0x00007f52753a864b: add    $0x10,%rsp
                       0x00007f52753a864f: pop    %rbp
 0.09%    0.02%        0x00007f52753a8650: test   %eax,0x167d19aa(%rip)  # 0x00007f528bb7a000
                                                                    ;   {poll_return}
                       0x00007f52753a8656: retq
```

Loop unrolling, up to -XX:LoopMaxUnroll

The difference



Return constant:
N != 100, N == 1

AND FINALLY

Best practices

- Small methods
- Megamorphic call-sites will be optimised if biased
- Controlled tests
- Look out for failure to inline
- Look out for de-opts
- Understand what is happening before attempting optimisation
- There is more than just the JVM at work...

Questions?

https://www.lmax.com/blog/staff-blogs

https://goo.gl/VQFupp

@epickrram

Thanks for the review:

Doug Hawkins

Nitsan Wakart